# INFORMATION TECHNOLOGY, COMPUTER SCIENCE, AND MANAGEMENT

UDC 004.42 https://doi.org/10.23947/2687-1653-2021-21-2-200-206

## Theoretical foundations of the organization of branches and repetitions in programs in the logic programming language Prolog

**D. V. Zdor**
Primorskaya State Academy of Agriculture (Ussuriysk, Russian Federation)

***Introduction.*** The organization of branches and repetitions in the context of logical programming is considered by an example of the Prolog language. The fundamental feature of the program in a logical programming language is the fact that a computer must solve a problem by reasoning like a human. Such a program contains a description of objects and relations between them in the language of mathematical logic. At the same time, the software implementation of branching and repetition remains a challenge in the absence of special operators for the indicated constructions in the logical language. The objectives of the study are to identify the most effective ways to solve problems using branching and repetition by means of the logic programming language Prolog, as well as to demonstrate the results obtained by examples of computational problems.

***Materials and Methods.*** An analysis of the literature on the subject of the study was carried out. Methods of generalization and systematization of knowledge, of the program testing, and analysis of the program execution were used.

***Results.*** Constructions of branching and repetition organization in a Prolog program are proposed. To organize repetitions, various options for completing a recursive cycle when solving problems are given.

***Discussion and Conclusions.*** The methods of organizing branches and repetitions in the logic programming language Prolog are considered. All these methods are illustrated by examples of solving computational problems. The results obtained can be used in the further development of the recursive predicates in logical programming languages, as well as in the educational process when studying logical programming in the Prolog language. The examples of programs given in the paper provide using them as a technological basis for programming branches and repetitions in the logic programming language Prolog.

***Keywords:*** logical programming, branching, repetition of predicates, recursive rule, recursion termination condition.

***For citation:*** D. V. Zdor. Theoretical foundations of the organization of branches and repetitions in programs in the logic programming language Prolog. Advanced Engineering Research, 2021, vol. 21, no. 2, pp. 200–206. https://doi.org/10.23947/2687-1653-2021-21-2-200-206



**Introduction.** Logical programming languages are used as tools for solving problems in building artificial intelligence systems [1]. One of the languages of non-procedural logic programming is Prolog. The program in this language uses the predicate calculus theory, it is a sequence of facts and rules that describe objects and define relationships between them. Then the goal is formulated, which is a statement that must be proved during the execution of the program. The program execution mechanism is based on the implementation of an attempt to prove the goal based on the facts and rules of the program using a standard backtracking and matching engine [2].

These circumstances significantly affect the approach used in the program preparation. In traditional procedural programming, the construction of the program is based on the algorithm for solving the problem. Procedural languages provide the software implementation of the compiled algorithm using operators. At the same time, structural procedural programming languages enable to implement basic algorithmic structures. Object-oriented programming is an evolutionary continuation of the development of this technology. The program is based on an object, its properties, methods and events. However, it should be noted that the event handler, which is a procedure executed when an event occurs in object-oriented programming languages, also contains a sequence of operators that solve a specific task. Here, in Prolog logic programming language, it is required to describe the problem and set the rules for its solution in the language of mathematical logic, taking into account the features of the program execution mechanism [3].

Thus, the composition of a program for solving a simple computational problem in Prolog will require the development of a separate approach related to the specifics of logic programming. Despite the fact that Prolog language has found its main application in the field of building expert systems, the software implementation of calculations is also a challenge, since computational tasks are included as elements of information processing systems, as well as intelligent ones [4].

A number of papers consider various aspects of programming in Prolog. The paper by Adam Lally and Paul Fodor describes the rules for matching with samples in Prolog. In particular, the authors suggest using backtrace instead of pattern matching, since it is required to check a lot of conditions during parsing, i.e., there is a need for a query language in which conditions can be included or excluded depending on some context. Prolog is recognized as an effective solution to the pattern matching problem, as well as the problems of depth-first search and backtracking. The researchers believe that despite its simplicity, Prolog is very expressive and allows recursive rules to represent reachability in parse trees, perform and treat the negation operation as a failure to check for absence of conditions [5].

N. I. Tsukanova examines in detail the subject area of using programming through logical models, demonstrates the connection of the basic logical concepts of predicates and Prolog basic language constructions. The paper discusses in detail the basics of logic programming and the program structure, basic algorithms, while using Visual Prolog 7 language as an example [6].

E. Costa considers Prolog as a logical programming language; he also describes in detail its main constructions, the program structure, the types of sentences on Prolog, the features of the program execution [7].

I. Bratko considers artificial intelligence algorithms in Prolog. The main value of the work is that the researcher demonstrates the use of Prolog in various areas of artificial intelligence, including for performing heuristic search, programming under constraints, machine learning, etc. [8].

The solution to logical problems in Prolog is studied in the paper of A. N. Adamenko. It is of fundamental importance, since the author considers recursion as a way of organizing the repetition of predicates [9].

In the paper of V. T. Tarushkin, P. V. Tarushkin, L. T. Tarushkina and A.V. Yurkov, the logic of predicates and Prolog is considered, which allows us to draw a number of valuable theoretical conclusions about the basics of using a program language [10].

O. P. Soldatova and I. V. Lezina consider Prolog as an element of logical programming and axiomatic systems. The paper also discusses the main strategies for solving problems, the procedural nature of the program, in particular, repetition and recursion of the language [11].

Michael A. Covington, Roberto Bagnara, Richard A. O'Keefe, Jan Wilemaker and Simon Price consider the problem of coding in Prolog. The researchers' work contains guidelines for code markup, naming conventions, documentation, proper use of Prolog functions, program development, debugging and testing. Each guide provides its justification, and where there are controversial points, illustrations of the relative pros and cons of each alternative are given [12].

V. N. Markov presents an overview of new and traditional logic programming tools, and also analyzes the main paradigms of functional programming, which is organically implemented in Visual Prolog 7.5 version. Basic methods of processing and further representation of arrays, branches, repetitions, graphs, are also considered [13].

In their paper, Gopal Gupta, Enrico Pontelli, Khayri A.M. Ali, Mats Carlsson and Manuel V. Hermenegildo present a comprehensive overview of the problems of parallel implementation of logical programming languages, as well as the most relevant approaches. The researchers focus on the problems that arise when running Prolog programs in parallel, in particular, the organization of branches and repetitions. The paper describes basic methods of parallelism and shared memory parallelism, as well as their combinations [14].

It is also necessary to note the paper of O. N. Polovikova, V. V. Shiryaev, N. M. Obrabin and L. L. Smolyakova, where the features of performing logical tasks in Prolog are analyzed, in particular, a state-generated approach to finding answers and validation procedure, during which repetitions are performed and branches are organized. The paper presents a solution to a logical problem, which in practice illustrates the approach developed by the researchers [15].

The analysis has shown that the literature sources consider the solution to computational problems in which it is required to implement branching or repetition according to the algorithm. In all the analyzed works there are direct or indirect descriptions of the organization of branches and repetitions in Prolog. However, there is no data on the maximum rationality of any approach, which indicates the incompleteness of knowledge about the use of the language and its capabilities when composing programs in the context of logical programming.

The study objective is to determine the most rational theoretical basis for the organization of branches and repetitions in programs in Prolog language.

**Materials and Methods.** The analysis of specialized literature on the research topic for the last 15 years has been carried out. Methods of comparative analysis, generalization and systematization of knowledge, the program testing, analysis of the program execution, were used.

**Research Results.** Based on the analysis of literary sources, the most rational method of organizing branches and repetitions in programs in Prolog is determined. The theoretical background for the organization of branches and repetitions in programs in this language is formulated. It should be noted that the process of proving the program purpose is reduced to comparing the statements included in the purpose to the facts and rules from the program knowledge base. The truth of the statements included in the purpose is established in the order of their sequence in the purpose from left to right. In this case, the comparison with the predicates of the program knowledge base is carried out in the order of their sequence from top to bottom. If at some step of the program execution, i.e., the proof of the problem solution, the comparison of some predicate fails, the Prolog system uses a second process — backtracking.

To control the search for a solution in Prolog, there is a number of standard predicates that provide changing the standard search mechanism. These predicates include the predicate fail, which has the value “false”, and the cut predicate, which can be written as the “!” sign.

When organizing a branch, it is required to construct it. Mutually exclusive conditions must be included as tail predicates in the rules that make up the branching structure. Let us consider an example of finding the largest of two numbers. It is known that the maximum of two numbers in mathematics can be determined as follows:

$$\max(x, y) = \begin{cases} x, & x \geq y \\ y, & x < y \end{cases}.$$

In Prolog, the finding of max (x, y) can be set as follows:

***predicates***

max (***real, real, real)*** /* triadicpredicate */

***clauses,***

max (X, Y, X) :- X > =Y.

max (X, Y, Y) :- X<Y.

The knowledge base does not contain facts, but contains two rules. The first rule can be read as follows: “Maximum of values X, У is X, if X > = Y”, the second rule: “Maximum of values X, У is У, if X < Y”.

If you set an external goal

***goal***

max (15, 10, Max). /* X =15, У =10 */

Prolog will find the solution:
Max = 15
1 solutions.

When finding a solution, Prolog needed only the first rule, because for given values X > У. Since the goal is external, Prolog will also check the second rule for finding another solution and spend time on it. The conditions in the rules are mutually exclusive, so if you add cut predicate: X > Y.

max (X, Y, X) :- X > =Y, !. or max (X, Y, Y) :- X <Y, !.
max (X, Y, Y) :- X <Y. max (X, Y, X) :- X > =Y.

Prolog, having found maximum value according to the first rule, will no longer check the second one. In this example, the presence of the cut predicate is due to the logic of the program.

Let us consider the ways of organizing repetitions in the program in Prolog language. To organize the repetition of actions, you need to set a cyclic construction:

***repeat.***
***repeat:- repeat.***

This construction sets an infinite loop. You can use a predicate with any name as a loop predicate. The predicate repeat is not standard in this case and should be described in the section predicates. The program must provide for an exit from the cyclic design.

We consider as an example a program for finding function values $y = x^2$ for arguments that the user enters sequentially from the keyboard:

***predicates***
tab
repeat
test(real)
***clauses***
repeat.
repeat:-repeat.
tab:-repeat, readreal(X), test(X).
test(10e10):-nl.
test(X):-Y=X*X, write("X=",X," Y=",Y), nl, fail.
***goal***tab.

If you enter numbers from the keyboard, the values of the argument will be displayed on the screen, and the calculated value of the function will be displayed next to it. The program execution will end if a number in the exponential form 10e10 is entered. Until the specified number is entered, the tail predicate test(X) will have the value "false" due to the presence of the predicate ***fail*** in the tail of the rule test(X). In this case, Prolog uses backtracking, and the predicate ***repeat*** loops the process on entering a new number. If you enter a number in exponential form 10e10, the goal will be provedıo

In this example, the organization of repetitions is implemented through a construction using the rule ***repeat:- repeat***, which, in fact, is recursive, since both the head and the body there is a predicate with the same name.

Now let us consider the organization of repetitions directly using a recursive rule on the example of the task of tabulating function $y = x^2$ on segment $[a;b]$ with step $h$:

***predicates***
tab(real,real,real)
***clauses***
tab(A,B,H):-A<=B, X=A, Y=X*X, write("X=",X," Y=",Y),nl,
A1=A+H, B1=B, H1=H, tab(A1,B1,H1).
***goal***

```
write("a="), readreal(A), write("b="), readreal(B),
write("h="), readreal(H), nl, tab(A,B,H).
```

When the program is executed, a table of function values will be displayed on the screen for a user-defined segment with a specified step.

This program contains recursive rule tab(A,B,H). To exit the recursion, the program uses the tail condition A<=B. tab (A, B, H). A<=B. The disadvantage of this method of exiting recursion is the fact that the predicate of the goal tab(A,B,H) after the program execution will have the value “false” because of the tail condition A<=B in the body of the recursive rule tab(A,B,H) after variable A reaches a value that exceeds the value of variable B. This disadvantage can be corrected by adding a non-recursive rule with the same name as the recursive one to the program. The program will take the following form:

```
predicates
tab(real,real,real)
clauses
tab(A,B,_):-A>B.
tab(A,B,H):-A<=B, X=A, Y=X*X, write("X=",X," Y=",Y),nl,
        A1=A+H, B1=B, H1=H, tab(A1,B1,H1).
goal
write("a="), readreal(A), write("b="), readreal(B),
write("h="), readreal(H), nl, tab(A,B,H).
```

Note that in this case, the recursive rule, which is responsible for repeating predicates, and the non-recursive rule, which is responsible for correct exit from recursion, use mutually exclusive conditions.

Here is another example of organizing repetition using a recursive rule. Let us consider a program for calculating the sum of k of the first terms of a series $S = 1 + x + x^2 + x^3 + \ldots + x^n + \ldots$:

```
domains
k=integer
x=real
n=real
s=real
predicates
sum(x,k,n,s)
clauses
sum(_,1,N,S):-N=1, S=N.
sum(X,K,N,S):-K1=K-1, sum(X,K1,N1,S1), N=N1*X, S=S1+N.
goal
write("X="), readreal(X),
write("K="), readint(K),
sum(X,K,N,S), nl,
write("S=",S).
```

In this program, k — the number of members of the series, whose sum must be calculated. Variable n is necessary to find the $i$-th term of the series, this variable contains the desired sum.

When trying to prove the statement of the goal, sum(X,K,N,S) the recursive rule sum(X,K,N,S) will be executed, provided that $K \neq 1$, as the tail condition of which there is recursive call sum(X,K1,N1,S1), in this case, the value of the variable is $K1 = K - 1$. So, the value of variable k will decrease to 1 during the direct course of recursion. After matching with the rule sum(_,1,N,S):-N=1, S=N the reverse recursion will start, at each turn of which the next term of the series will be calculated by multiplying the previous term by the value of argument x. The value of the sum of the series will be accumulated in variable s.

Thus, the organization of repetitions in the program in Prolog is implemented using recursion. A recursive rule sets an infinite loop. This loop is used to solve a computational problem. Here, the completion of the loop and the exit from the recursion can be implemented in different ways:

— input of a predefined variable value by the user from the keyboard;

— using a logical expression as one of the conditions of a recursive rule;

— using a non-recursive rule with the same name as the recursive one, but true statements must act as predicates of the body in this rule.

**Discussion and Conclusions.** The most effective ways of organizing branches and repetitions in the logic programming language Prolog, identified as a result of a comparative analysis of literary sources, are presented. The given tasks are for educational and training purposes. Their solution contributes to the understanding of the logical programming theory based on the application of analogy, taking into account the specifics of building a program in Prolog and a special mechanism for its execution.

The given examples of programs allow us to use them as a technological basis for programming branches and repetitions in Prolog. The obtained results can be used in the further development of the application of recursive predicates in logical programming languages, as well as in the educational process when studying logical programming in Prolog.

**References**

1. Adir E, Almog L, Fournier E, et al. Genesys-Pro: Innovations in Test Program Generation for Functional Processor Verification. IEEE Design & Test of Computers. 2014;21(2):84–93. https://doi.org/10.1109/MDT.2004.1277900

2. ARM Architecture Reference Manual. ARM DDI 0487A.f. ARM Corporation, 2015. 5886 p.

3. Kent D. Lee. Foundations of Programming Languages. Springer, 2017. 370 p.

4. Ute Schmid. Inductive Synthesis of Functional Programs: Universal Planning, Folding of Finite Programs, and Schema Abstraction by Analogical Reasoning. Springer Science & Business Media, 2013.

5. Adam Lally, Paul Fodor. Natural Language Processing with Prolog in the IBM Watson System. Association for Logic Programming, 2011.

6. Tsukanova NI, Dmitrieva TA. Teoriya i praktika logicheskogo programmirovaniya na yazyke Visual Prolog 7 [Theory and practice of logic programming in Visual Prolog 7]. Moscow: Hot Line – Telecom, 2013. 232 p. (In Russ.)

7. Eduardo Costa. Visual Prolog 7.3 for Tyros. 2010. 270 p. URL: http://visual-prolog.com/download/73/books/tyros/tyros73.pdf

8. Bratko I. Algoritmy iskusstvennogo intellekta na yazyke Prolog [Artificial intelligence algorithms in the Prolog language]. 3rd ed. Moscow: Williams; 2004. 637 p. (In Russ.)

9. Adamenko AN, Kuchukov AM. Logicheskoe programmirovanie i Visual Prolog [Logic programming and Visual Prolog]. Saint Petersburg: BKhV-Peterburg, 2003. 982 p. (In Russ.)

10. Tarushkin VT, Tarushkin PV, Tarushkina LT, et al. Logika predikatov i yazyk Prolog [Predicate logic and the Prolog language]. Modern High Technologies. 2010;4:62–63. (In Russ.)

11. Soldatova OP, Lezina IV. Programmirovanie na yazyke PROLOG [Programming in the PROLOG language]. Samara: Samara University Repository; 2008. 52 p. URL: http://repo.ssau.ru/handle/Metodicheskie-ukazaniya/Programmirovanie-na-yazyke-PROLOG-Elektronnyi-resurs-metod-ukazaniya-k-lab-rabotam-53131?mode=full (accessed: 20.06.2021). (In Russ.)

12. Michael A. Covington, Roberto Bagnara, Richard A. O'Keefe, et al. Coding guidelines for Prolog. Theory and Practice of Logic Programming. 2011;12(6):889–927.

13. Markov VN. Sovremennoe logicheskoe programmirovanie na yazyke Visual Prolog 7/5. [Modern logic programming in Visual Prolog 7/5]. Saint Petersburg: BKhV-Peterburg; 2016. 541 p. (In Russ.)

14. Gopal Gupta, Enrico Pontelli, Khayri A.M. Ali, et al. Parallel Execution of Prolog Programs: a Survey. ACM Transactions on Programming Languages and Systems. 2011;23(4):472.

15. Polovikova ON, Shiryaev VV, Oskorbin NM, et al. Osobennosti programmnoi realizatsii logicheskikh zadach na yazyke PROLOG [Features of software implementation of logical tasks in PROLOG]. Izvestia of Altai State University. 2021;1(117):116–120. URL: https://cyberleninka.ru/article/n/osobennosti-programmnoy-realizatsii-logicheskih-zadach-na-yazyke-prolog (accessed: 24.05.2021). (In Russ.)

Received 01.04.2021
Revised 14.04.2021
Accepted 04.06.2021

*About the Author:*

**Zdor, Dmitrii B.,** associate professor, Engineering Institute, Primorskaya State Academy of Agriculture (44, Bluhera St., Ussuriysk, RF, 692510), Cand.Sci. (Pedagogy), associate professor, ORCID: https://orcid.org/0000-0003-1131-6708, jevgeniya.999.gn@mail.ru

*The author has read and approved the final manuscript.*